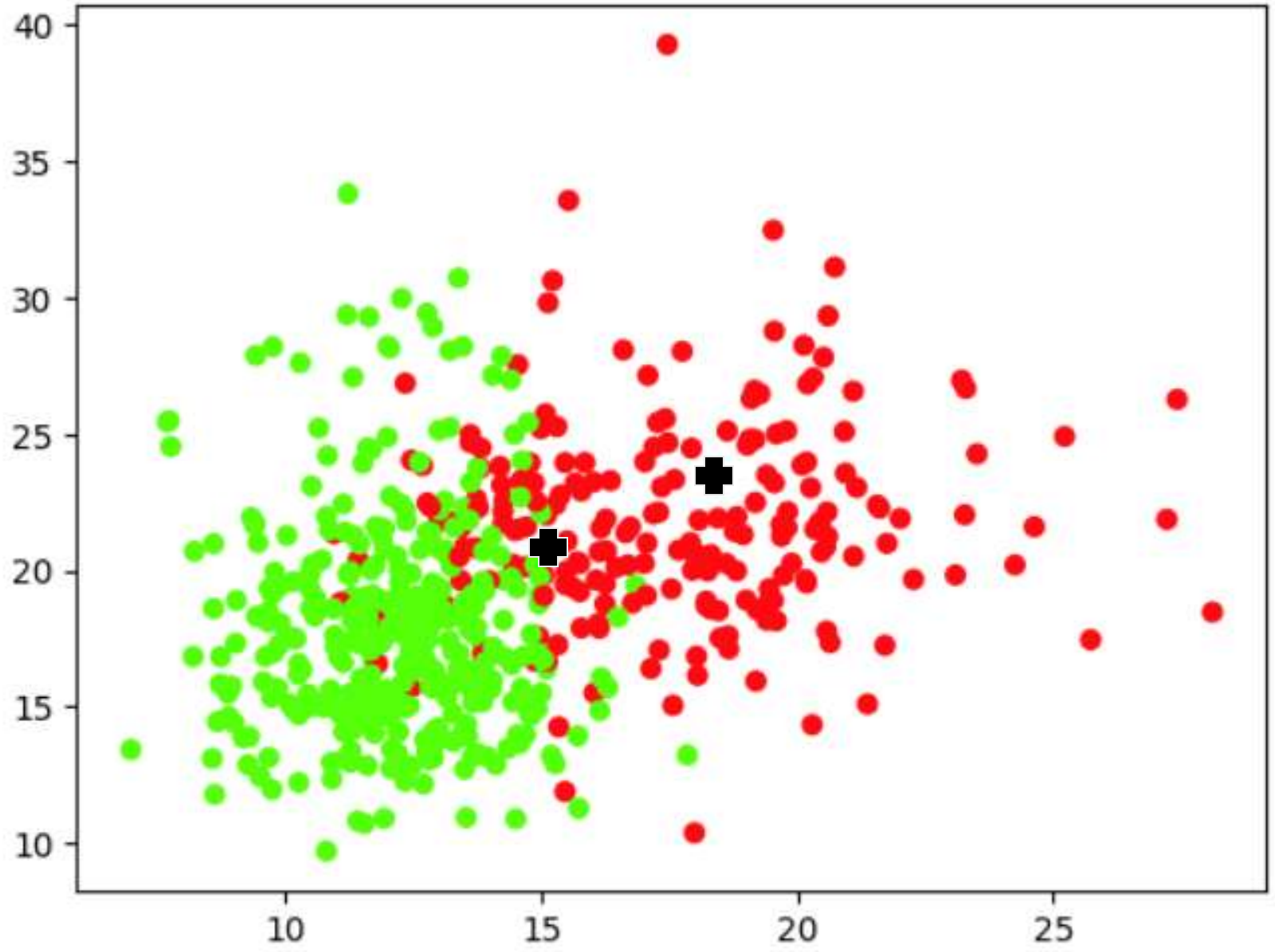
Nearest Neighbors

13 January 2026

Alex Lyman

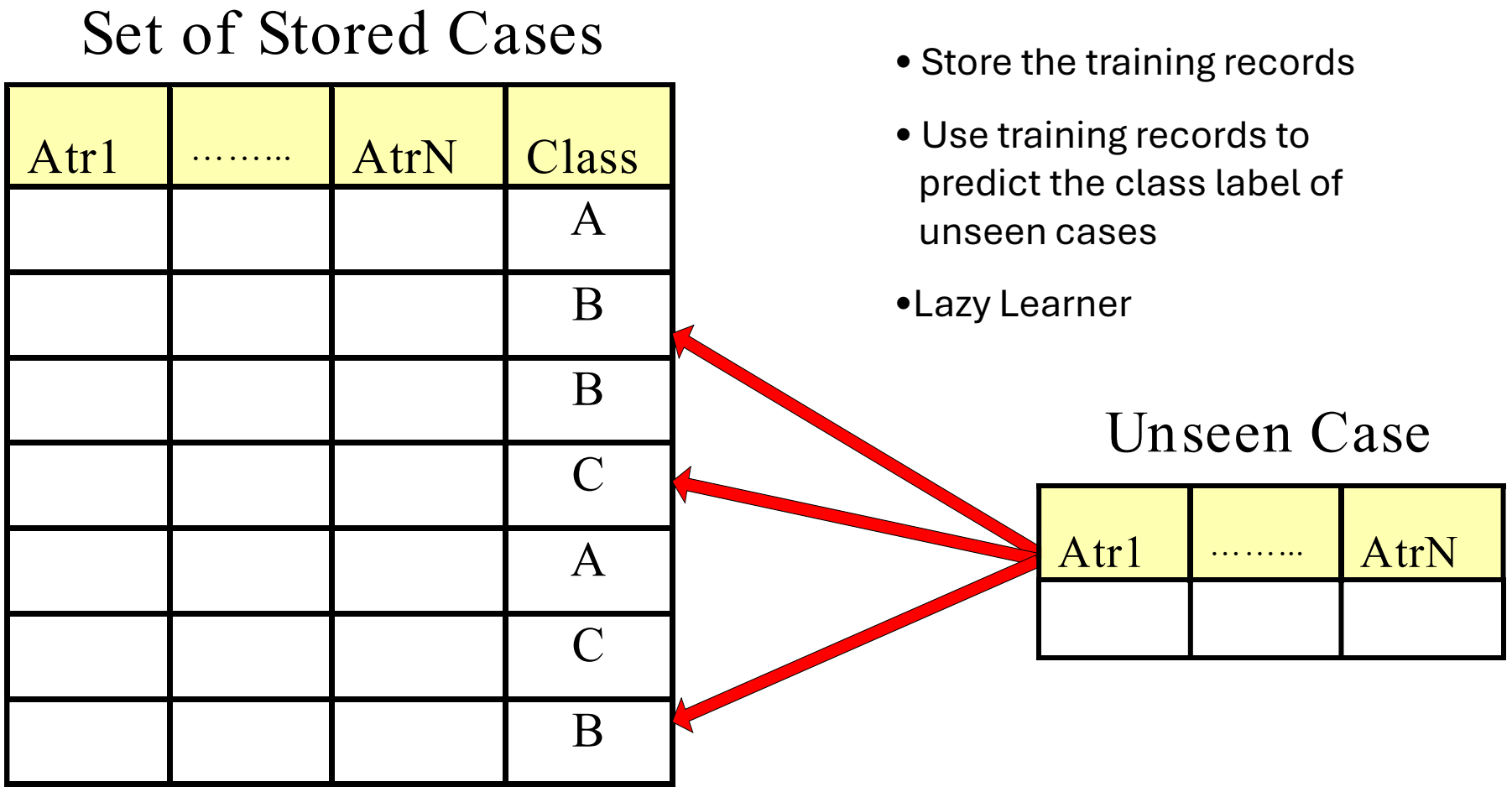
Nearest Neighbor Classification



How should we classify this instance?

And this one?

Instance-Based Classifiers



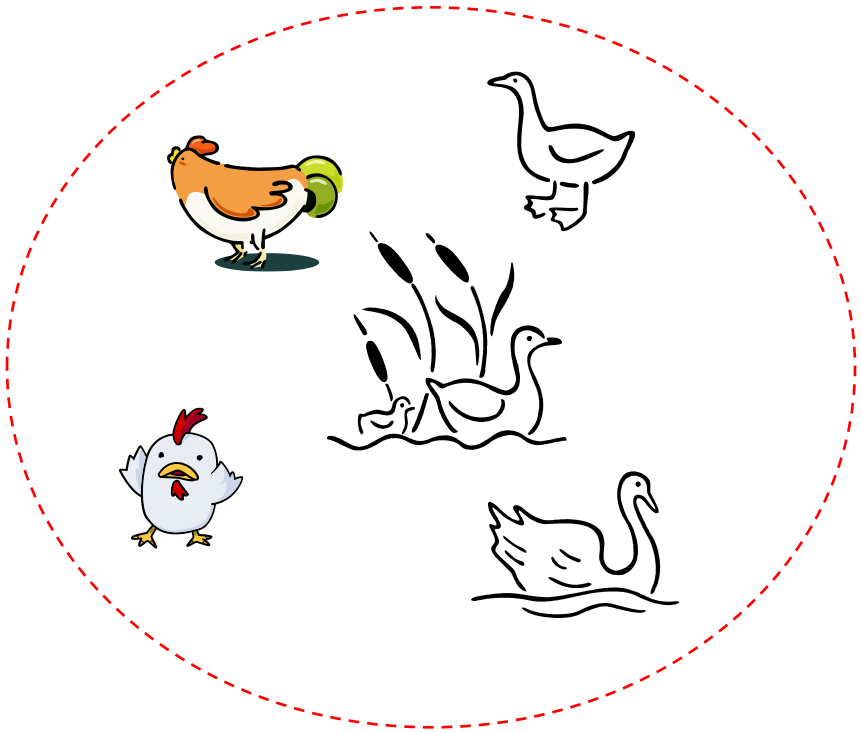
- Store the training records
- Use training records to predict the class label of unseen cases
- Lazy Learner

Instance Based Classifiers

- Examples:
  - Rote-learner
    - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
  - Nearest neighbor
    - Uses k “closest” points (nearest neighbors) for performing classification

Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck and quacks like a duck, then it's probably a duck



Training Records

Test Record

1-Nearest Neighbor

Task: predict the target / label of a new data point

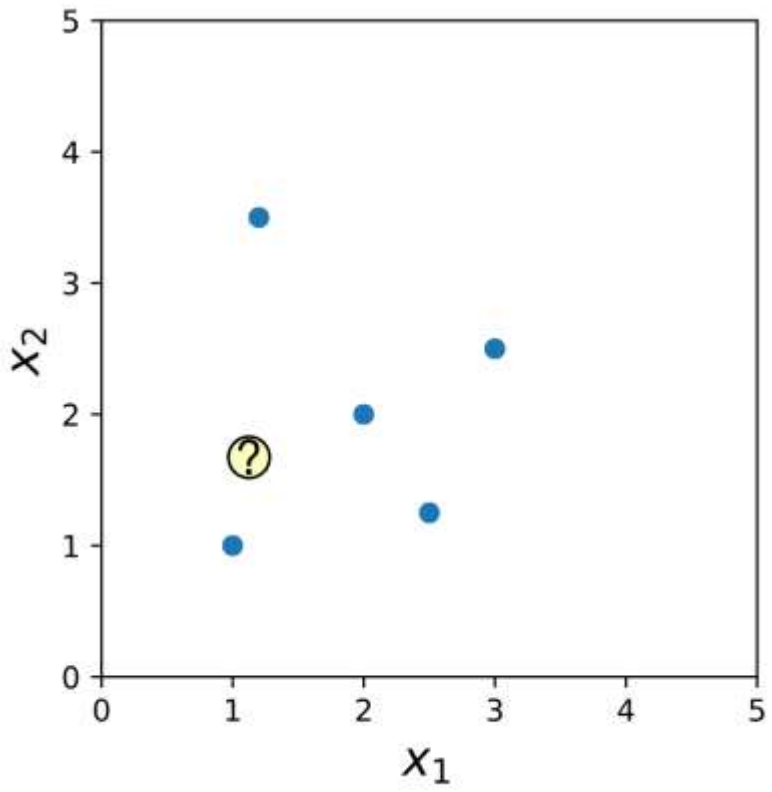
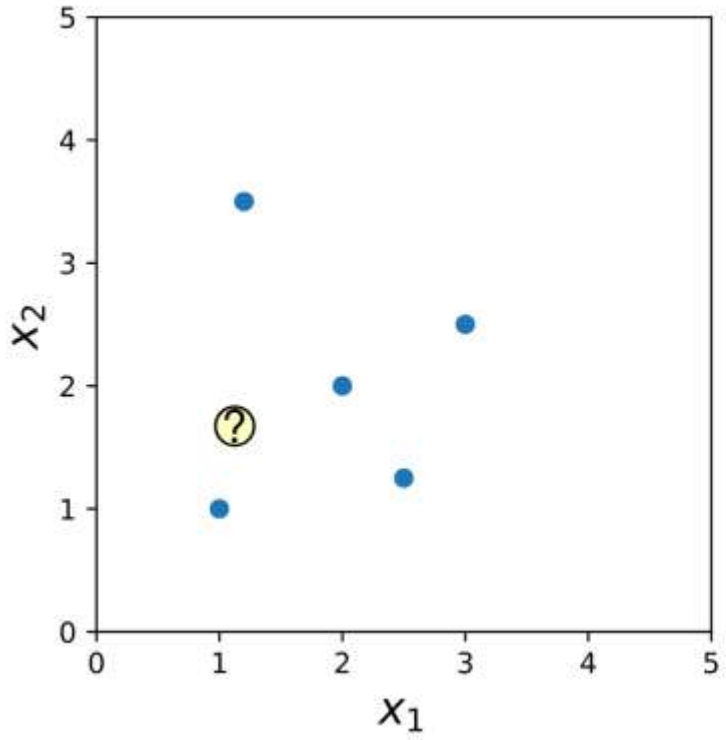


Image Credit: Sebastian Raschka

1-Nearest Neighbor

Task: predict the target / label of a new data point



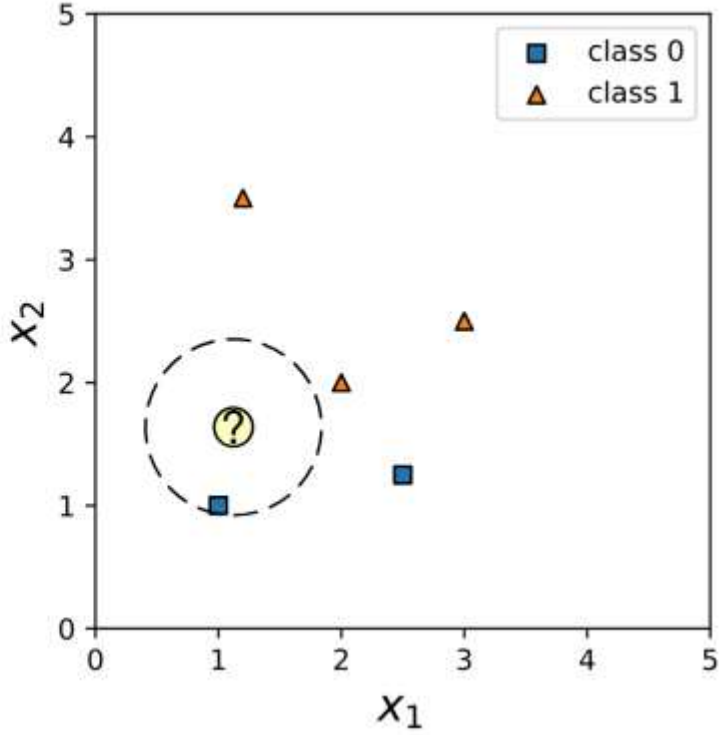
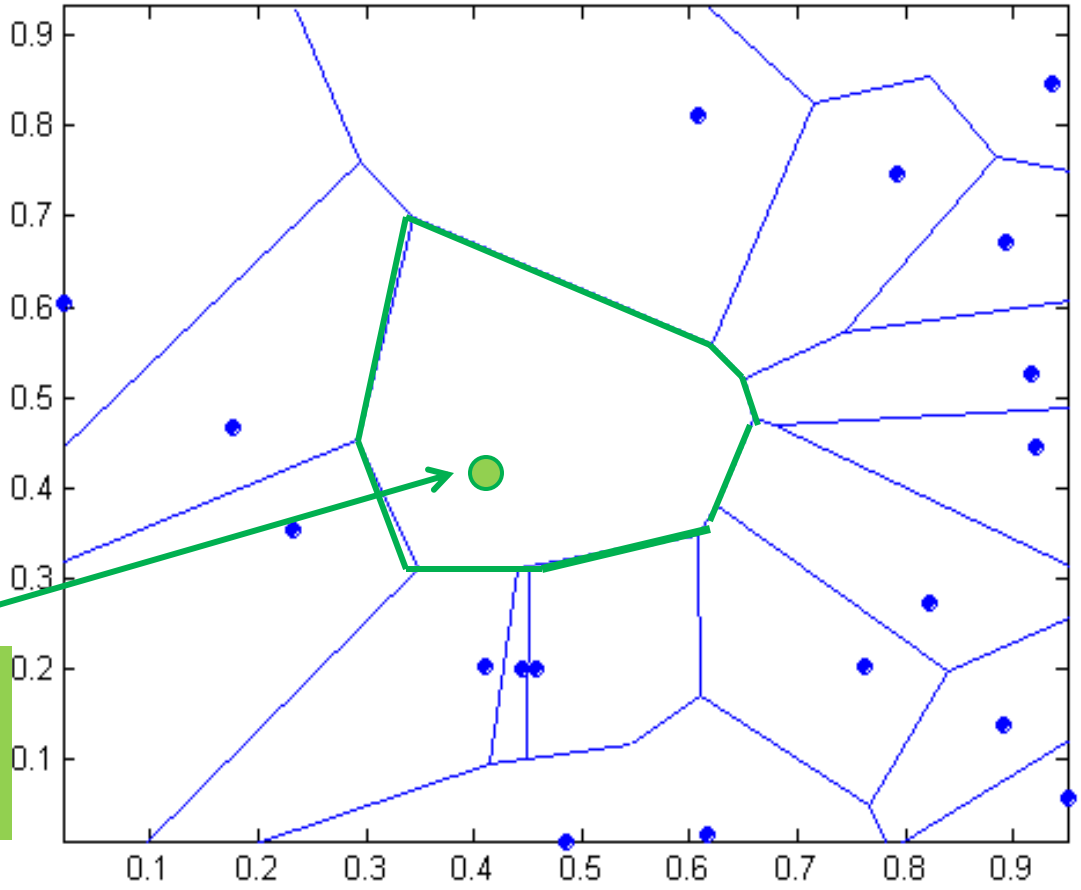


Image Credit: Sebastian Raschka

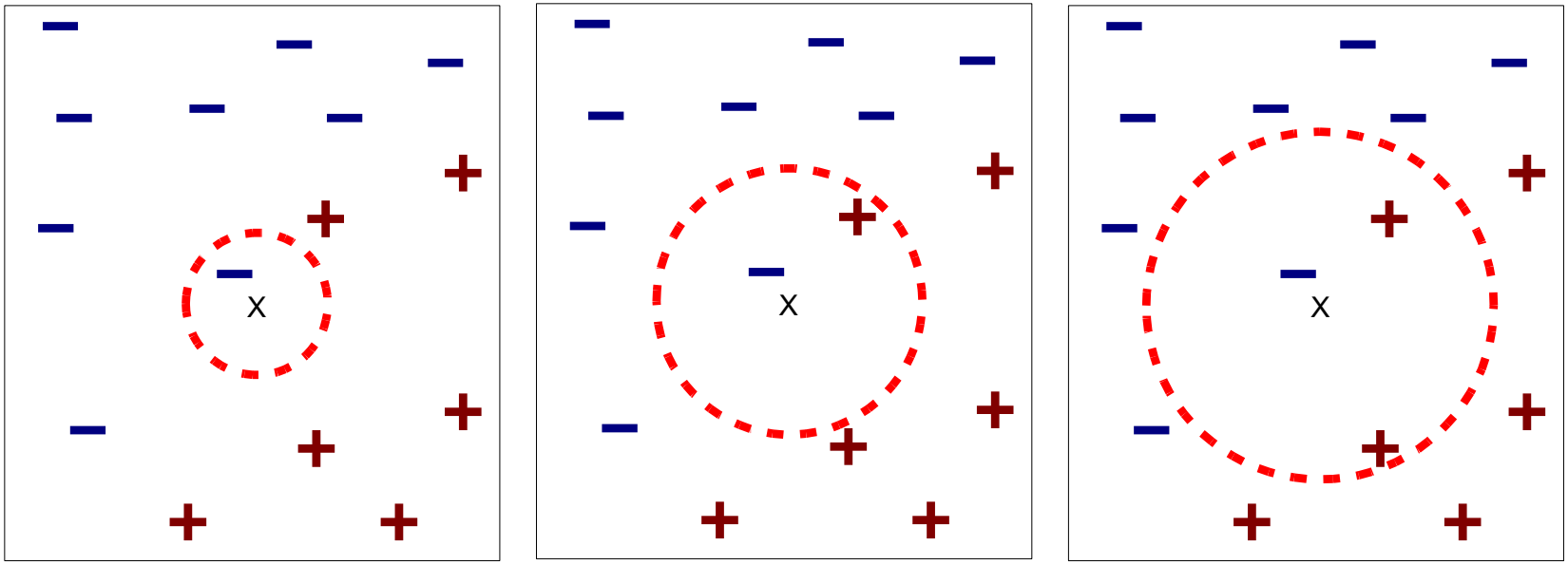
1 nearest-neighbor Boundry

Voronoi Diagram defines the classification boundary



The area takes the class of the green point

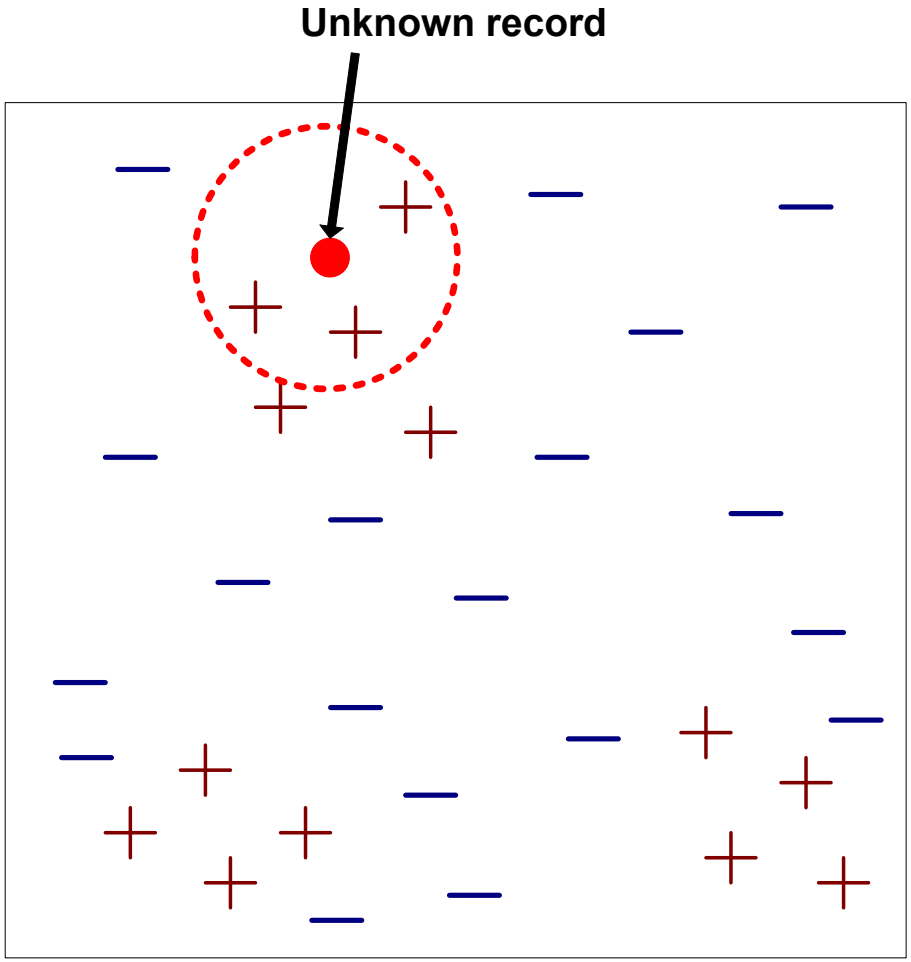
More than 1-Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

- K-nearest neighbors of a record *x* are data points that have the k smallest distances to *x* (the target point)
- K is a *hyperparameter* that we choose (giving inductive bias)

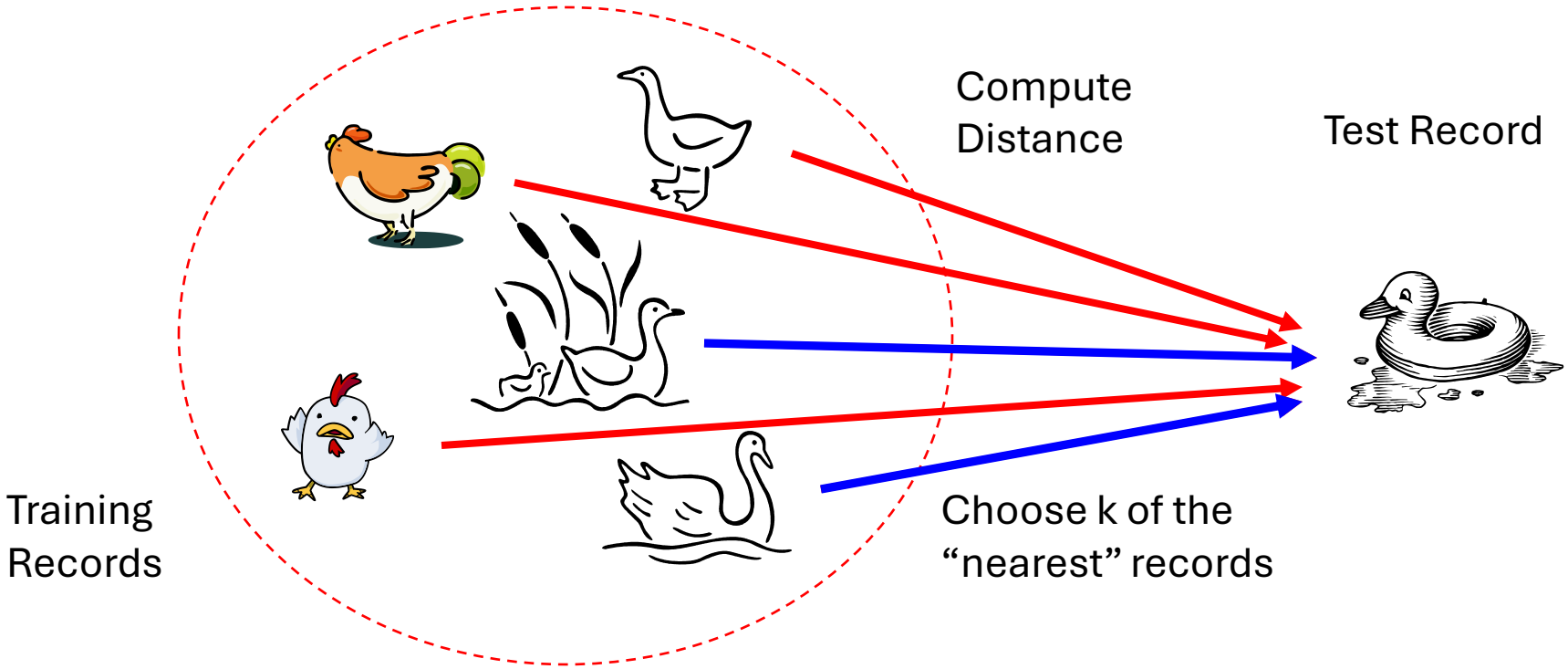
K-Nearest-Neighbor Classifiers



- Requires three things
  - The set of stored records (coordinates of points)
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve (hyperparameter)
- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of target record (e.g., by taking majority vote)

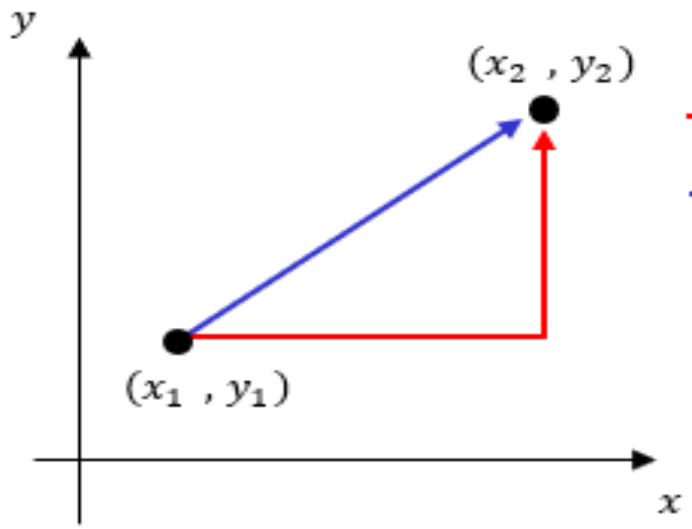
Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck and quacks like a duck, then it's probably a duck



Nearest Neighbor Classification

- Compute distance between two points:
  - Euclidean distance or Manhattan distance typically



Manhattan Distance L^1

Euclidean Distance L^2

$$L^1 = |x_2 - x_1| + |y_2 - y_1|$$

$$L^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- What is the Manhattan distance between (0,0) and (1,1)?
- What is the Euclidean distance between (0,0) and (1,1)?

Nearest Neighbor Classification

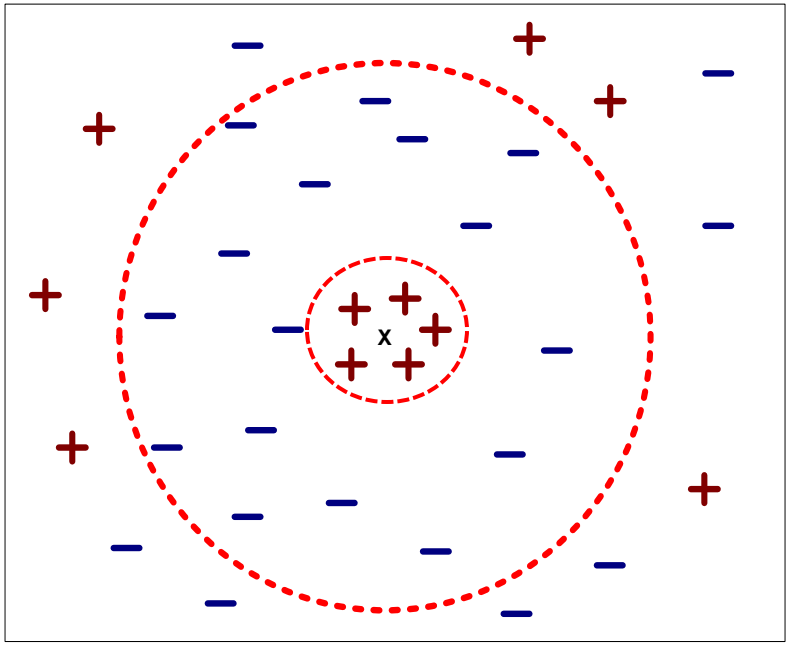
- Compute distance between two points:
  - Euclidean distance (Pythagorean Theorem in 2d)

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
    - (ties, k odd or even)
  - **Weigh** the vote according to distance
    - weight factor, $w = 1/d^2$

Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



Nearest Neighbor Classification…

- Scaling issues
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 90lb to 300lb
    - income of a person may vary from $10K to $1M

    - What happens when you calculate distance?
    - From last time

Standardization

–Transform values into number of standard deviations from the mean

$$new\ value = \frac{current\ value\ - average}{standard\ deviation}$$

Normalization

–Transform values to fall within a specified range

$$new\ value = \frac{current\ value\ - \min value}{range}$$

–Often, range = max value – min value => [0, 1]

Nearest neighbor Classification…

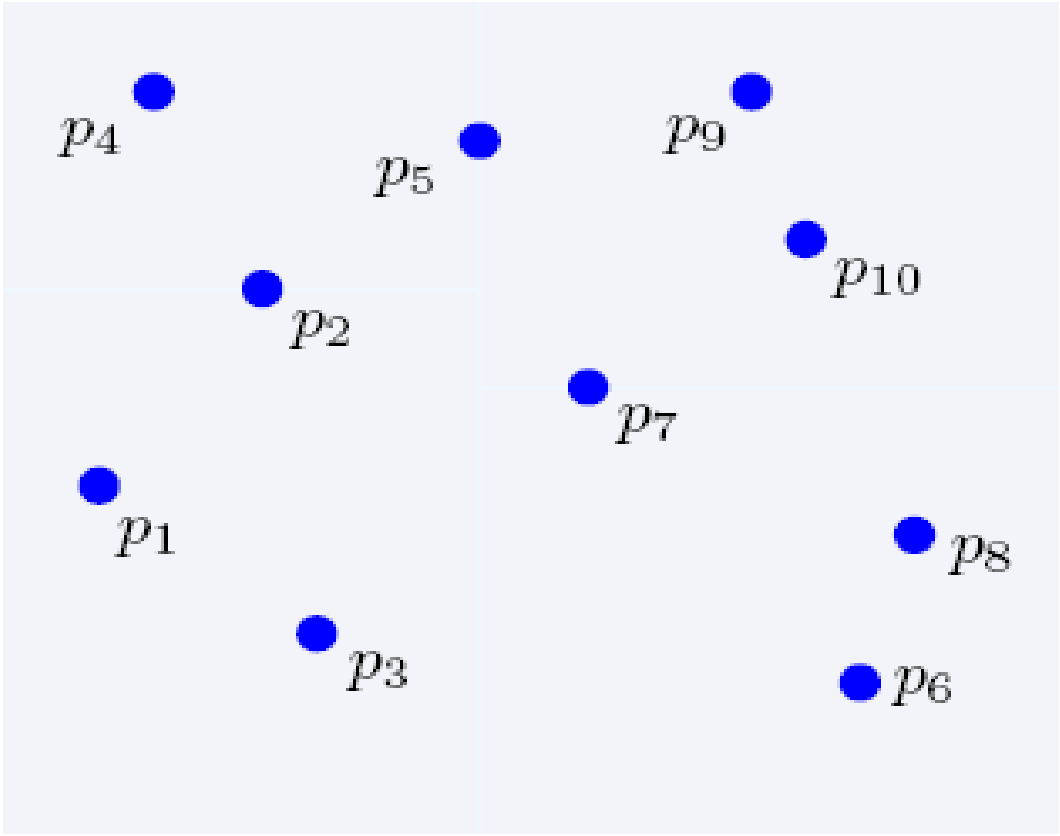
- k-NN classifiers are lazy learners
  - It does not build models explicitly
  - Unlike eager learners such as decision tree induction and rule-based systems
- Classifying unknown records is relatively expensive
  - Naïve algorithm: O(n)
  - Need for structures to retrieve nearest neighbors fast.
    - The Nearest Neighbor Search problem.

Nearest Neighbor Search

Nearest Neighbor Search

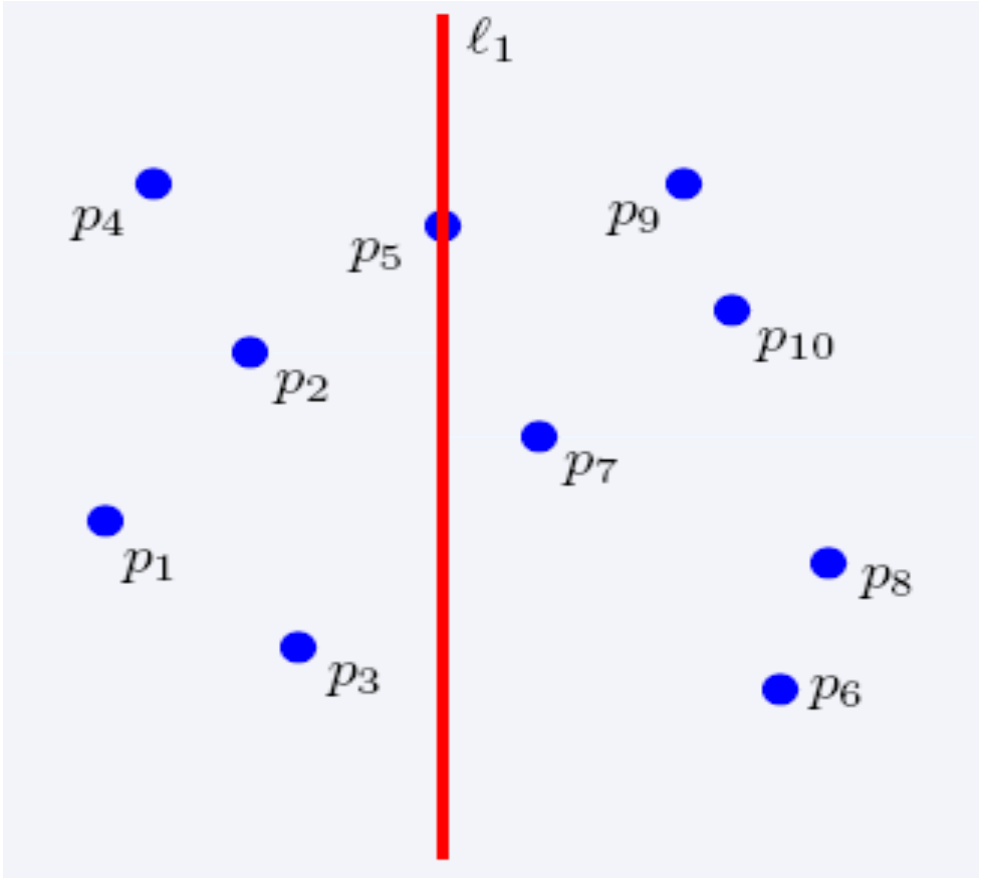
- Two-dimensional kd-trees
  - A data structure for answering nearest neighbor queries in R^2
- kd-tree construction algorithm
  - Select the x or y dimension (alternating between the two)
  - Partition the space into two with a line passing from the median point
  - Repeat recursively in the two partitions as long as there are enough points
- Only efficient in low-dimensional space

Nearest Neighbor Search



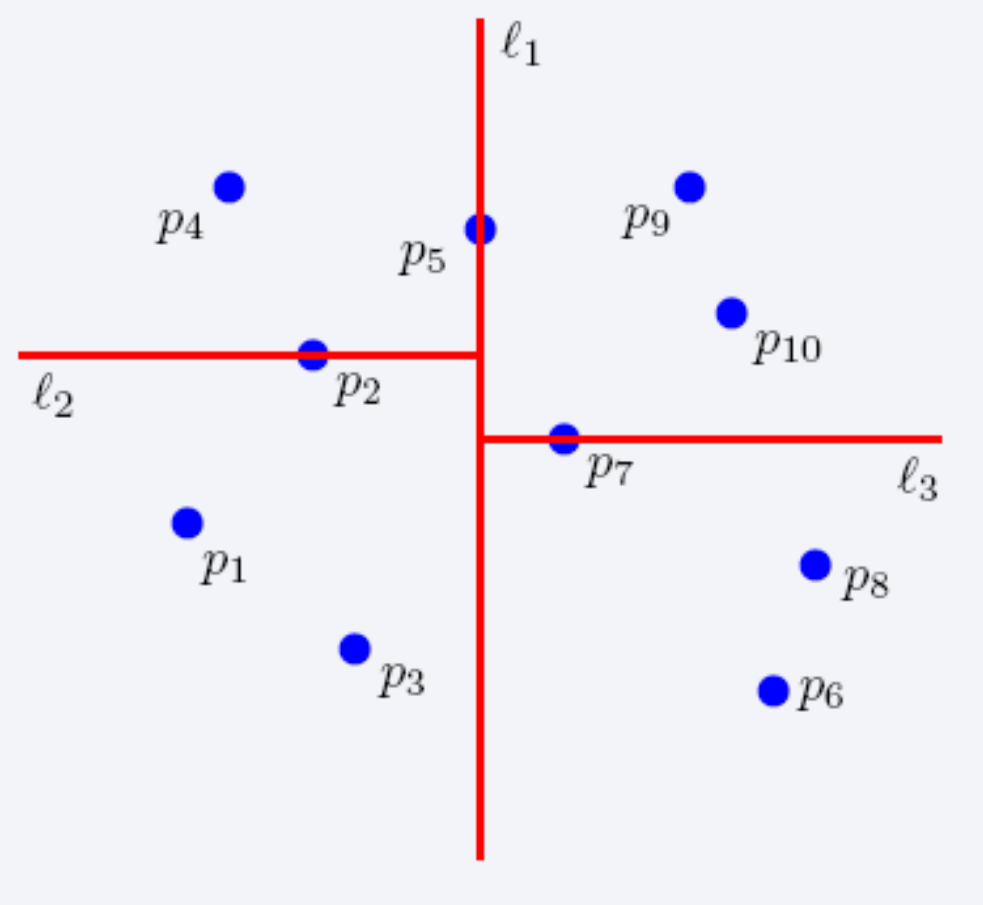
2-dimensional kd-trees

Nearest Neighbor Search



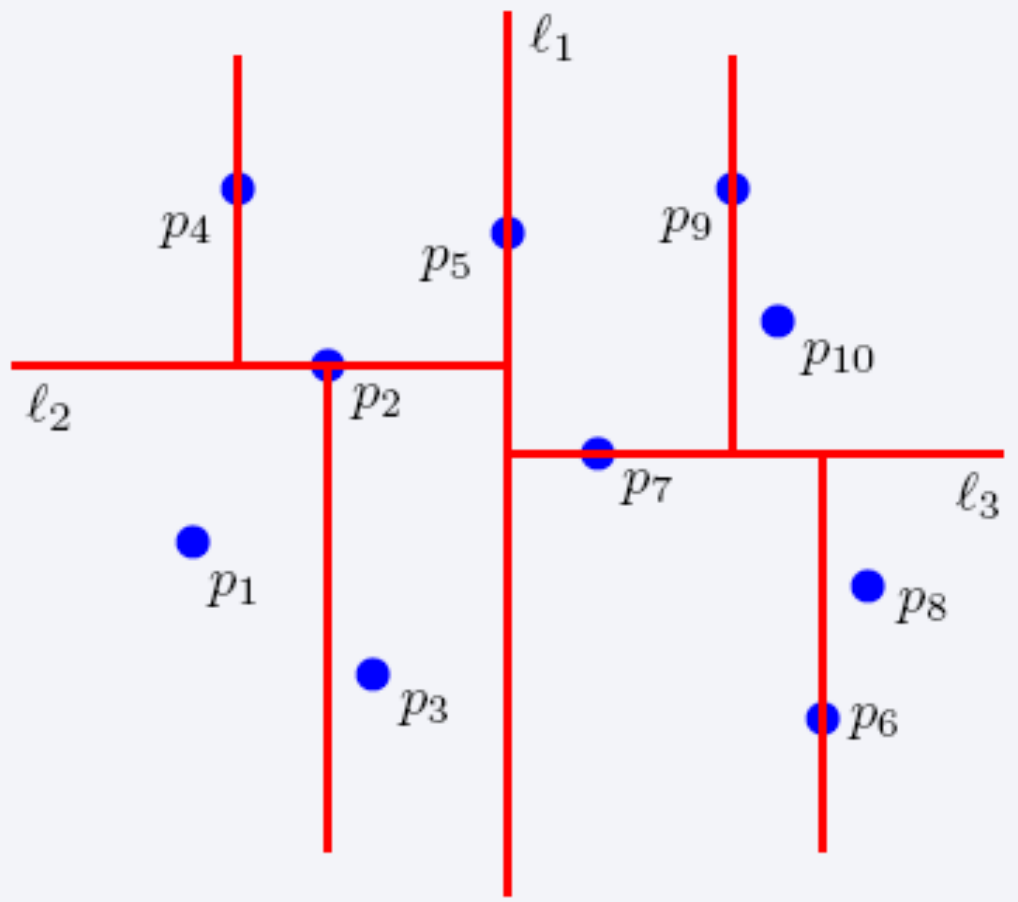
2-dimensional kd-trees

Nearest Neighbor Search



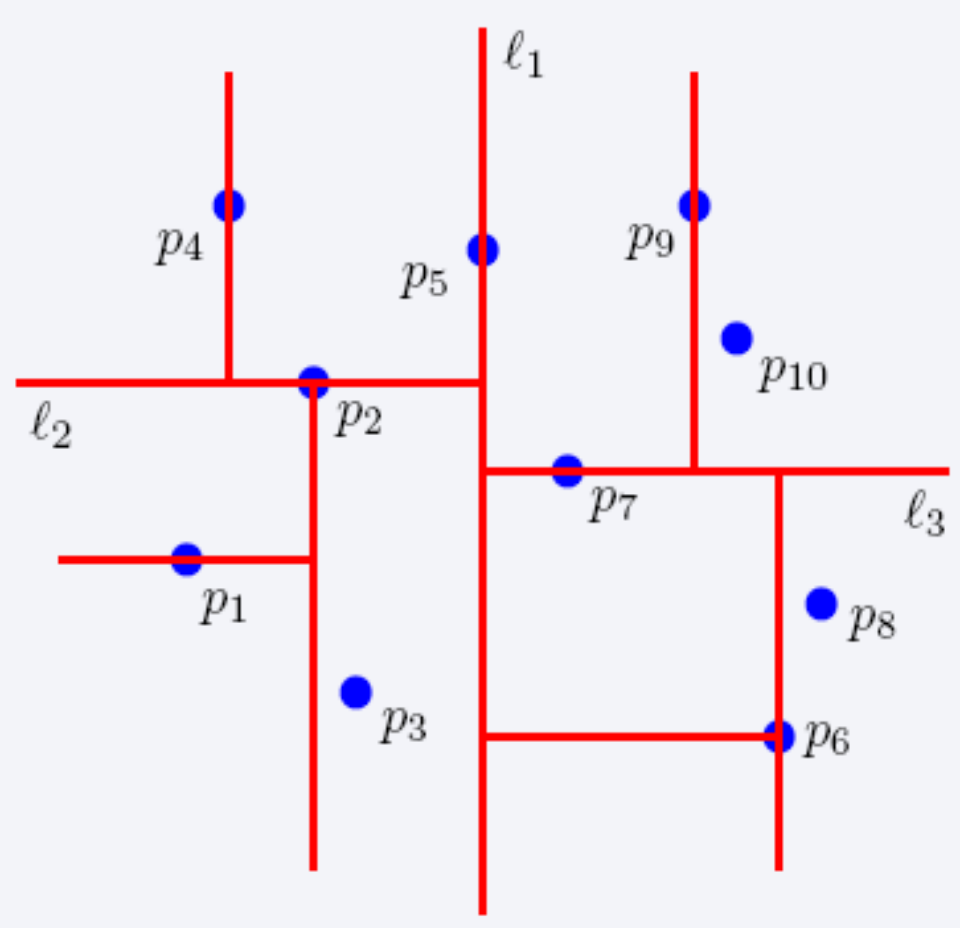
2-dimensional kd-trees

Nearest Neighbor Search



2-dimensional kd-trees

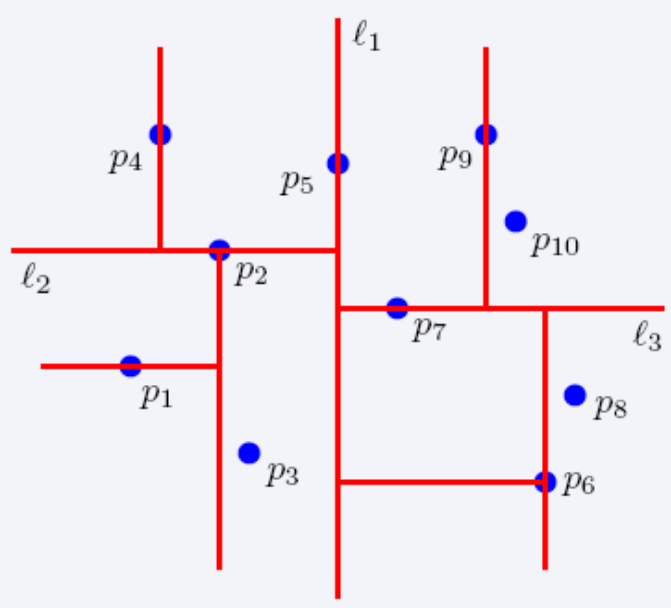
Nearest Neighbor Search

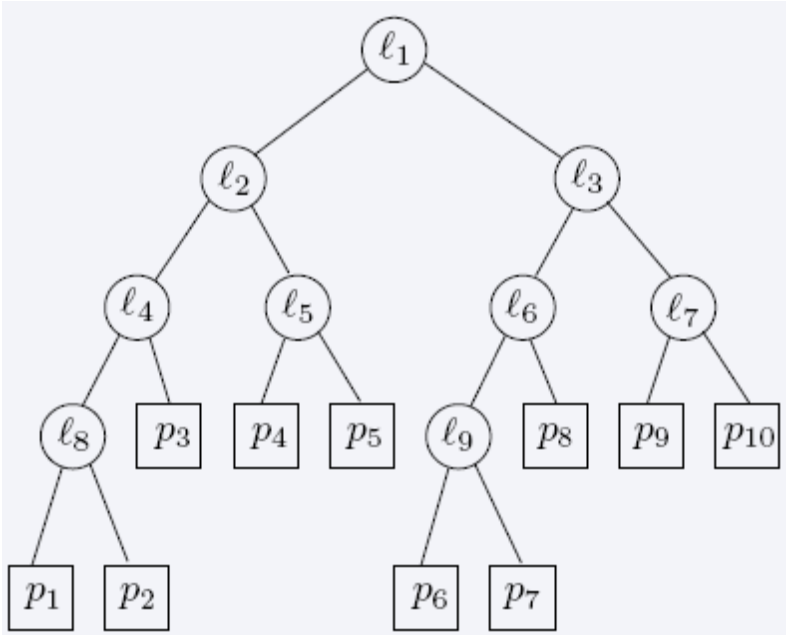


2-dimensional kd-trees

Nearest Neighbor Search

2-dimensional kd-trees

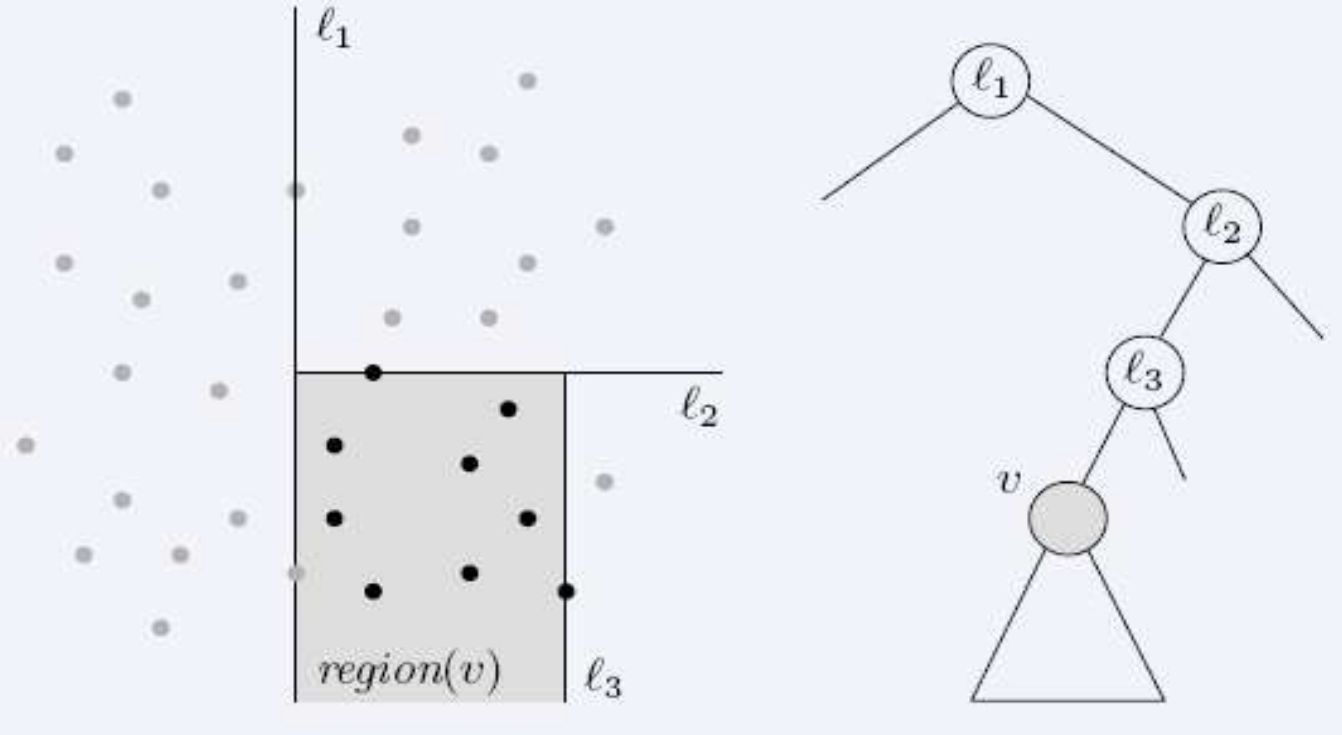




Nearest Neighbor Search

2-dimensional kd-trees

region(u) – all the black points in the subtree of u



Nearest Neighbor Search

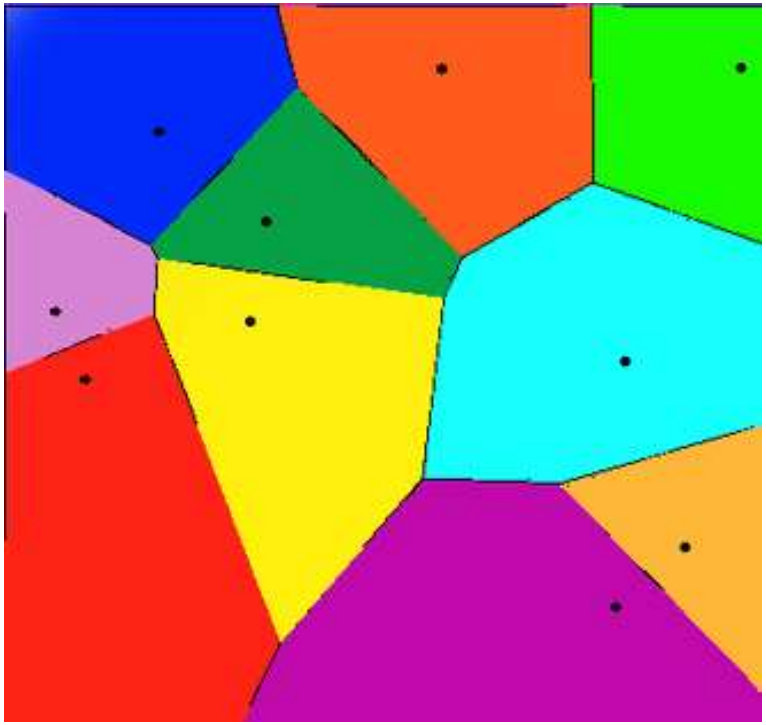
2-dimensional kd-trees

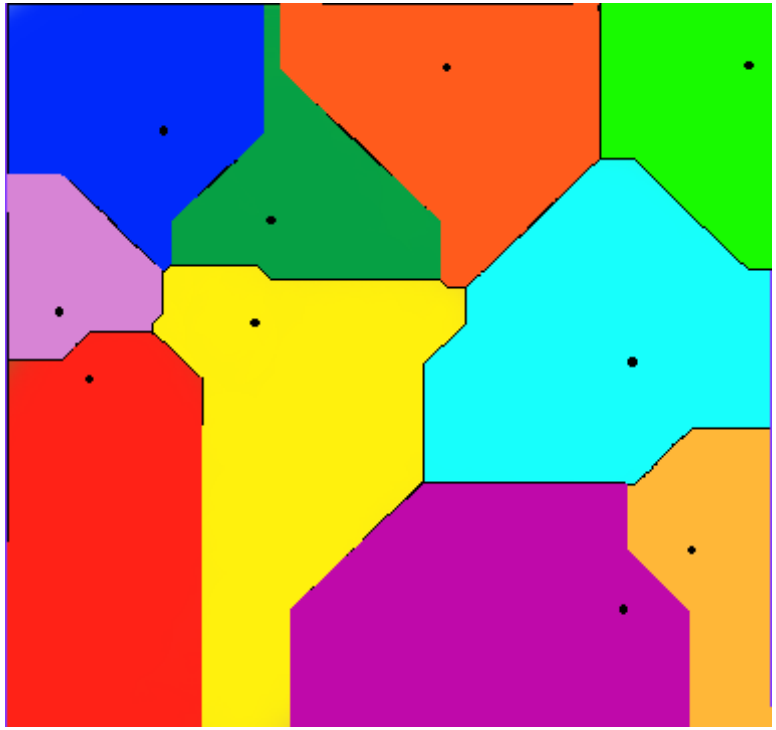
- A binary tree:
  - Size **O(n)**
  - Depth **O(logn)**
  - Construction time **O(nlogn)**
  - Query time: worst case **O(n),** but for many cases **O(logn)**

Generalizes to d dimensions

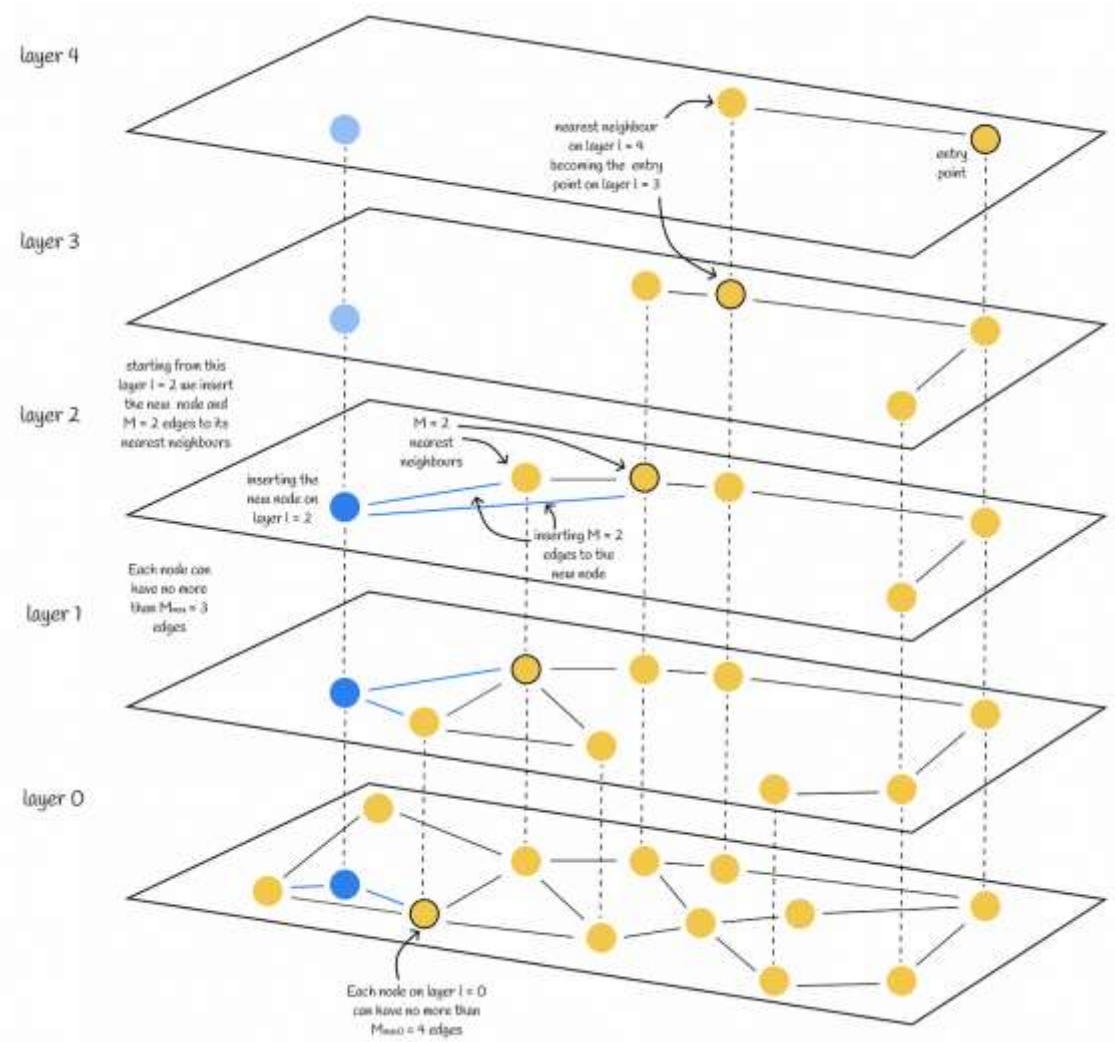
Nearest Neighbor Search

- Can also use the Voronoi diagram
  - Generalizes to d dimensions
  - Need to generalize for k nearest neighbors





Hierarchical Navigable Small Worlds



Used in vector databases

approaches Log N

outperforms KD-tree and brute force

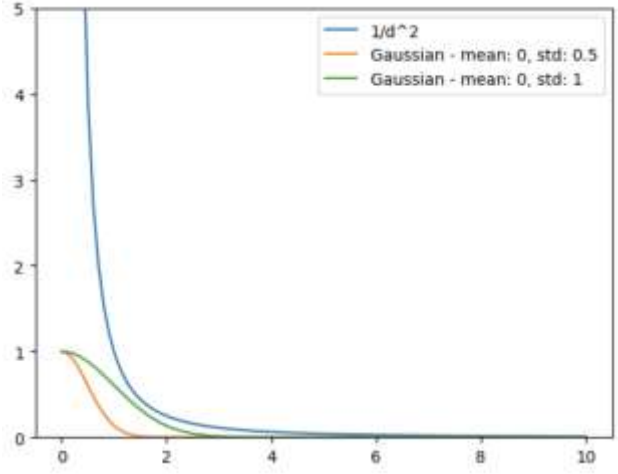
You don’t need to know how this works.

k-Nearest Neighbor (cont)

- Usually do distance weighted voting where each nearest neighbor vote is scaled inversely to its distance
- Inverse of distance squared is a common weight

$$w_i = \frac{1}{dist(x_q, x_i)^2}$$

- Gaussian is another common distance weight
- In this case the *k* value is more robust, could let *k* be even and/or be larger (even all points if desired), because the more distant points have negligible influence
- *w* = 1 for the non-weighted version



Quiz Time!

Challenge Question - *k*-Nearest Neighbor

- Assume the following data set
- Assume a new point (2, 6)
  - For nearest neighbor distance use Manhattan distance
  - What would the output be for 3-nn with no distance weighting? What is the total vote?
  - What would the output be for 3-nn with squared inverse distance weighting? What is the total vote?

$$L^1 = |x_2 - x_1| + |y_2 - y_1|$$

A. A A

B. A B

C. B A

D. B B

E. None of the above

x	*y*	*Label*
1	5	A
0	8	B
9	9	B
10	10	A

$$w_i = \frac{1}{dist(x_q, x_i)^2}$$

Challenge Question - *k*-Nearest Neighbor

- Assume the following data set
- Assume a new point (2, 6)
  - For nearest neighbor distance use Manhattan distance
  - What would the output be for 3-nn with no distance weighting? What is the total vote? – B wins with vote 2 out of 3
  - What would the output be for 3-nn with distance weighting? What is the total vote? A wins with vote .25 vs B vote of .0625+.01=.0725

x	*y*	*Label*	*Distance*	*Weighted Vote*
1	5	A	1 + 1 = 2	$1/2^2 = .25$
0	8	B	2 + 2 = 4	$1/4^2 = .0625$
9	9	B	7 + 3 = 10	$1/10^2 = .01$
10	10	A	8 + 4 = 12	$1/12^2 = .0069$

Attribute Weighting

Attribute Weighting

- Normalize Features!

- One of the main weaknesses of nearest neighbor is irrelevant features, since they can dominate the distance
  - Example: assume 2 relevant and 10 irrelevant features
- Most learning algorithms weight the attributes
  - MLP and Decisions Trees do higher order weighting of features

- Could do attribute weighting - No longer lazy evaluation since you need to come up with a portion of your hypothesis (attribute weights) before generalizing

- Still an open area of research
  - Higher order weighting – 1st order helps, but not enough
  - Even if all features are relevant features, all distances become similar as number of features increases, since not all features are relevant at the same time, and the currently irrelevant ones can dominate distance
  - An issue with all pure distance based techniques, need higher-order weighting to ignore *currently* irrelevant features
  - Dimensionality reduction can be useful (feature pre-processing, PCA, NLDR, etc.)

Value Difference Metric (You don't need to know this super well, just that it exists)

- Distance of two attribute values is a measure of how similar they are in inferring the output class

- Assume a 2-output class task (A, B)
- Attribute 1 = Shape (Round, Square, Triangle, etc.)
- 10 total round instances
  - 6 class A and 4 class B
- 5 total square instances
  - 3 class A and 2 class B
- Since both attribute values suggest the same probabilities for the output class, the distance between Round and Square would be 0
  - If triangle and round suggested very different outputs, triangle and round would have a large distance

Value Difference Metric (VDM)

[Stanfill & Waltz, 1986]

Providing appropriate distance measurements for nominal attributes.

$$vdm_a(x,y) = \sum_{c=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2$$

$N_{a,x}$ = # times attribute a had value x

$N_{a,x,c}$ = # times attribute a had value x and class was c

C = # output classes

Two values are considered closer if they have more similar classifications, i.e., if they have more similar correlations with the output classes.

VDM Example

$$vdm_a(x,y) = \sum_{c=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2$$

$N_{a,x}$ = # times attribute a had value x
$N_{a,x,c}$ = # times attribute a=x and class was c
C = # output classes

Color →	Class
red	0
green	0
red	1
blue	0
blue	1
red	1
blue	1
blue	1

x	$N_{1,x}$	class	$N_{1,x,c}$	$N_{1,x,c}/N_{1,x}$
red	3	0:	1	1/3=.33
		1:	2	2/3=.67
green	1	0:	1	1/1=1.0
		1:	0	0/1=0.0
blue	4	0:	1	1/4=.25
		1:	3	3/4=.75

vdm_{color}(red,green) = 0.889
vdm_{color}(red,blue) = 0.014
vdm_{color}(green,blue)=1.125

I got these values

0.8978
0.0128
1.125